A software method of emulation of EEPROM memory

The object of the invention is a software method of emulation of the EEPROM memory in another non-volatile memory, for example the Flash type memory. This method is applicable in systems, where in order to decrease costs of devices, using a non-volatile EEPROM memory, the existing memory is used, for example the Flash type memory for emulation of the EEPROM memory. The solution includes integration of the method with circuits of the device and memory management methods, for example monitoring of changes and management of data writing. An exemplary format of data writing in the emulated memory was also presented for the needs of the application.

There is a hardware method of memory emulation in the integrated circuit itself, known from the European patent application number EP0991081. Its disadvantage is that it is limited to one circuit and type of memory and a low ratio of the size of emulated EEPROM memory to the size of the Flash memory used for the same purpose (1:8) and a small size of the emulated memory, up to 8KB. The disclosed solution works with every currently applied Flash memory circuit, and it can emulate any size of currently produced EEPROM circuits, up to 64KB.

Another hardware solution is the circuit, described in American patent document no. US 5,651,128 presenting a memory, which consists of a matrix of cells and circuits, which enable programming of deletion of the emulated memory, the Flash type memory is used in the example of embodiment of the emulation.

In case of the described solutions, the emulation of the EEPROM memory is made with the use of a hardware solution, where the circuit uses the Flash type memory to store data both assigned for saving in the Flash memory as well as in the EEPROM memory.

The difference between the presented methods and the solution according to the invention consists in that the emulation according to the invention, in contrary to the known solutions is conducted in a software way. Thanks to it, this method can be applied in any device and in any type of the Flash memory. The essence of the invention is the logical software layer, which controls monitoring and use of the emulated memory. Moreover, by selecting this type of emulation both the costs of purchase of memory circuits and the use of the surface of printed circuit boards are lower, eliminating, much more expensive than the Flash memory (in calculation per 1 KB of memory), the EEPROM memory circuit, which is generally assembled as a separate module. Additionally the benefit is based on the decreased, simplified and miniaturized electronic circuit, which hitherto was using the EEPROM memory.

In the described solution according to the invention, an exemplary format of data writing in the emulated memory is also presented.

The example of digital television decoder, described below, where the solution according to the invention is applied, should be treated as one of possible applications of the emulation method of the EEPROM memory according to the invention.

Each device, which requires a non-volatile memory of the first type, for example a the Flash type memory and the EEPROM type memory, can be designed in such a way that the method of emulation of the EEPROM memory in the storage of the first type, according to the invention, is used and thus both the costs of circuits and the quantity of the required space for digital circuits assembled on the printed circuit boards are decreased. This allows for designing universal devices, the key element of which is independence from configuration of the memory block, where a combination of the Flash and the EEPROM memory or a combination of the Flash and emulated EEPROM memory can appear.

The elimination of the EEPROM memory is usually possible without a need to increase the size of the FLASH memory. The increase of the Flash memory is more favorable than the cost of an additional the EEPROM circuit, in view of the price of the EEPROM memory, which has 32 times higher price per 1KB in relation to the FLASH memory. The additional Flash memory can serve not only for the emulation of the EEPROM memory.

One of the problems, which are encountered with emulation of the EEPROM memory in the Flash type memory, is the fact that the Flash memory operates in a different way. Data should be changed by whole sectors. They cannot be changed by bytes, like in case of the EEPROM memory, which forces the use of a driver, which, by making operation available according to typical the EEPROM memories, will operate on sectors of the Flash memory. In order to decrease the number of required write operations to the Flash memory another solution was applied. Because during a typical work of the EEPROM memory, data are updated frequently and in small quantities, for example with a single byte, during emulation of the EEPROM memory data are collected and saved after a certain time as a

patch. Such time can be, for example, defined in seconds or as a number of changes made on the data stored in the operational memory. Additionally, the saved data can be compressed if that is favorable. One of the requirements for operation of the emulation of non-volatile EEPROM memory is to guarantee possession of a correct copy of data even if these are not the most up-to-date data.

Such situation takes place for example in case of a voltage failure during the operation of data writing. In order to ensure the required security of data of the emulated EEPROM memory, servicing of a data copy was applied. That is why the system requires a double size of space in the Flash memory in order to emulate a given size of the EEPROM memory. After programming data in one of the two sectors of the Flash memory, the second sector must be erased. In case of emulation of the EEPROM memory with a size of 32KB, two sectors of the Flash memory are used, with the size of 64Kb each. Additionally, there are three types of buffers in the operational RAM memory. The first of them stores always the most up-to-date image of the emulated EEPROM memory. The second one stores the last patch, and the last one is optionally used for storing the patch after compression.

The invention is illustrated in the example of embodiment in the drawing, in which fig. 1 shows an exemplary device in the form of a digital television decoder using the method according to the invention. Fig. 2 discloses the procedure of starting memory emulation system, fig. 3 – the procedure of saving data to the memory, fig. 4 – an exemplary format of the patch. Fig. 5 presents a sector of the Flash memory including the data of the emulated EEPROM memory and patches, fig. 6 pictures the format of the header of the patch, fig. 7 – the procedure of saving update data, while in turn fig. 8 shows the procedure of preparing data for

updating with a possibility of canceling the previously saved patch, and fig. 9 – the procedure of selecting the current non-volatile memory sector.

The signal receiver, presented in fig. 1 of the drawing, which is a decoder of digital television, is presented for the needs of the invention in a simplified version, with only these elements disclosed, which are required for presenting the idea of the invention. The decoder of digital television **101** includes many modules. The most important of them is the processor **120** which manages operation of the device. Additionally, according to the invention, the processor services an internal block **121**, which controls emulation of the EEPROM memory and a signal-processing block **122**. There is a signal from the signal receiving block **110** connected to the processor. Additionally, the processor has a possibility of bidirectional exchange of data through external interfaces **140**. The digital television decoder includes also a few types of memory, which are bidirectionally connected with the processor. These are for example, a non-volatile memory, favorably of the Flash type **150**, and operational RAM memory **160**. There are programs stored in these memories, which control the operation of the digital television decoder. Blocks **130** and **131** make it possible to transmit the output A/V signal respectively and communicate with external control devices, for example a remote control unit (RCU).

Fig. 2 presents a process of initiating emulation of the EEPROM memory emulation according to the invention. This procedure is performed at starting the device, which uses emulation. The procedure starts in the point **201**. Next in step **202**, a sector is selected, from which data will be read, as the current sector. One of two sectors is selected on the basis of a few criteria. They will be presented in details in fig. 9. Next, in step **203** of the procedure, the content of the second sector is erased. Point **203** of the procedure is executed, if the auxiliary memory sector contains data, despite that it should not. Such situation can take place at

the first start of the emulation of the EEPROM memory, it there were other data earlier in this memory sector or in case, when saving of the first patch of data update to the second was interrupted, for example as a result of a failure of power supply, or just after finishing it, but before deletion of the content of the first sector. In such case, at the next start of emulation, there are data in two sectors and that is why one of them is erased. Next reading of selected sector **204** of the Flash memory is initiated, fetching the original image of the emulated EEPROM memory into the operational RAM memory. It can be for example 32KB of one sector of the Flash memory, the size of which is usually 64KB. If there is an error during reading, for example data are not correct; the content of the current sector is erased. In the point **205** of the procedure the first patch of data update is collected from the Flash memory and its validity is checked, i.e. it is checked if it contains valid data **206**. If the patch was not invalidated (the point **805**), in the point **207** of the procedure, it is checked if the patch data are compressed. If they are compressed, in the point **208** it is decompressed. In opposite case the procedure goes directly to the point **209**, where data of the patch are saved in the operational memory RAM, storing the current image of the emulated EEPROM memory. The last point of the procedure is to check **210** if there are still data to be read in the Flash memory. If it is so, the procedure processes the next patches according to the described algorithm. If the last patch is saved, the procedure finishes its operation. After completing the procedure from fig. 2, there are all data, of the emulated EEPROM memory, available in the operational memory. Fig. 3 illustrates the procedure of saving data of the EEPROM memory, being emulated, in the Flash memory. It starts in the point **301**, where data are prepared for updating the content of the Flash memory. These data are also saved in the buffer of the RAM

operational memory, which always stores the current contents of the emulated EEPROM memory, which will be presented in detail in fig. 8 of the drawing.

Next, in step **302** of the procedure, a check is made if the size of the patch is bigger than the defined value. These are 64 bytes in the presented example of embodiment. If it is so, the procedure moves to the point **303**, where the patch is being compressed. In the example of embodiment it is assumed that the patch, the size of which is lower than 64 bytes, is not being compressed due to a low probability of a reduction of the patch size. Next, in the point **304** of the procedure the result of compression from the point **303** is checked. This check defines if the gain from data compression is large enough, to bear the additional cost of time needed for decompression of patch at the moment of initiating the system of emulation of the EEPROM memory according to the invention. If it is so, the compressed patch is processed further, and if not, the uncompressed patch is being processed. Next the saving procedure moves to the point **305**, where it is checked if there is appropriate space to write a new patch of updating data in the current sector of flash memory. If it is so, the patch, i.e. the recently modified data, is saved in the point **306** in the Flash memory – which is in detail illustrated in fig. 7 in connection with fig. 6. In opposite case, the current sector is changed into the second one in the point **307** of the procedure. Next the full image of operational RAM memory stored in the buffer, i.e. the current image of the emulated EEPROM memory is saved as a new original image of the emulated EEPROM memory in the point **308** of the procedure. Next in the point **309** the content of the second sector of the Flash memory is erased, because only then it is certain that the data of the emulated EEPROM memory will not be lost. In this place in the newly selected sector of the Flash memory there is already additional place for new patches.

The procedure of saving finishes its operation in the point **310**.

An exemplary format of the patch, in accordance with which data are saved in the Flash memory, is presented in fig. 4. It consists of four fields, out of which the first one is a header of the patch **401**, the second is the size of the data group **402** (the field, which appears in case of patches, with many groups of data update (so called Multi Block Patch), the third one is an offset of data in relation to the initial address **403**. This field appears only in case of uncompressed patches. The last field are the data **404**. The patch can include many groups of data (multi block patch), of which every one is saved under a different memory address. In case when the patch is compressed, it does not contain an offset field, while the value of the offset itself is read only after decompression of the patch. In case when the patch contains many data groups (multi block patch), the values of fields **403** contain an offset in relation to the final address of the previous data group. In this way only the first offset defines the absolute address, and next values are relative addresses in relation to the previous data group. This allows for decreasing the number of bits, where memory addresses are saved.

Division of the Flash memory sector into two parts was illustrated in fig. 5. The first of them **501** is the original image of the emulated EEPROM memory, and the second **502** is a set of patches of this memory. A sector in the Flash memory has a size of 64KB, and therefore it always contains a full original image of the emulated EEPROM memory, even uncompressed, as well as up to a few thousands of patches. The full image is saved at the beginning of each sector as the original image of the emulated memory. A full image can be saved every time and the sector can be changed with every record, but the solution with the application of patches is more effective, because these patches are generally much smaller than 32KB.

The format of the header of the patch from fig. 4 was presented in fig. 6. The elements **601** – **604** correspond to **401** – **404**. The header **601** consists of 7 parts **601a** – **601g**. The bit **601a** is the start bit, changed at the time of starting preparation for saving the patch. **601b** is the bit of "correct writing of the size and format" changed after the size and format of data are correctly saved. **601c** is a bit of "correct writing", changed after the whole patch is correctly saved in the Flash memory. By means of these three bits during data reading **206** one can ascertain how many consecutive bytes could have been changed in the Flash memory and based on this define where the next patch can be located. If, at any time, there is a failure of power supply, the next patches can be saved without a necessity of deleting the whole sector, right after the patch, saving of which was not finished. **601d** is the bit, which is changed after the patch is invalidated. If thus, the next patch would restore the state of the EEPROM memory from before the last update, instead of saving the next patch, the previous one can be invalidated. **601e** are two bits, the value of which defines the format of the patch. By means of two bits one can define four formats, however 3 formats can be defined in the exemplary solution, and the fourth possible value can be reserved for a future upgrade of the system, according to the idea of the invention. The exemplary formats are:

- 0x00 a single uncompressed block
- 0x01 a single compressed block
- 0x10 many groups of uncompressed data

The last field of the header of the patch is **601f**, in which the total quantity of data is defined in the patch, regardless of the size of the header. In case of fields **601f**, **602** and **603** the first two bits define the format, in which the values of data

quantity and address offsets are saved. The values of the bits for the field **602b** can determine for example if:

- 0x00 the field is described with the use of 4 bits.
- 0x01 the field is described with the use of 8 bits.
- 0x10 the field is described with the use of 12 bits.

The values of these bits for fields **601f** and **603b** can determine for example if:

- 0x00 the field is described with the use of bits from the current byte.
- 0x01 the field is described with the use of bits from the current and next byte.
- 0x10 the field is described with the use of bits from the current byte and the next two bytes.

Fig. 7 presents in detail the point **306** of the procedure from fig. 3 of the drawing. Writing of the patch starts in the point **701**, where the value of the start bit **601a** is changed. Next in the point **702** of the procedure the size and type of the patch are saved – fields **601e**, **601f**. If an error occurs, the procedure ends. In majority of cases an error is a power supply failure. If saving of the values of fields is correct, the procedure moves to the point **703**. In this step the value of the bit in the field **601b** is changed. Next, in the point **704** of the procedure, separate data groups are saved. If an error occurs, the procedure ends. If saving is correct, the procedure moves to the point **705**. In this step, the value of the bit in the field **601c** is changed. At this moment the procedure ends, and the patch is correctly saved in the Flash memory. After saving there is an additional possibility of invalidating the patch. If the patch is to be invalidated, in the procedure illustrated in fig. 8 the value of the field **601d** is changed. The procedure of preparing data for updating with a possibility of invalidating the previously saved patch is shown in fig. 8. The diagram is a particularization of the point **301** from fig. 3 of the drawing. The proce-

dure starts in the point **801**, where the patch is prepared for saving in the memory. The patch header is being created among others. Next in the point **802** data are saved in RAM memory, which stores the most current image of the EEPROM memory, at the same time the prepared patch is stored in the buffer of the RAM memory. In turn, in step **803** of the procedure a check is made, if the patch recently saved in the Flash memory is valid – the value of the field **601d**. If the patch is invalidated, the writing process **807** of a new patch is continued. In opposite case, when the hitherto saved patch is valid, the procedure advances to point **804**, where it is checked if the currently processed patch reverses changes introduced with the saving of the previous patch. If not, the writing process of the patch is continued **807**. In the opposite case, when the patch reverses changes introduced by saving of the previous patch, the procedure advances to the point **805**. In this place the value of bit **601d**, which invalidates the previously saved patch, is changed. Further the procedure moves to the point **806**, where it cancels (stops) saving of the new patch to the Flash memory.

The procedure of selecting the current sector is presented in detail in fig. 9. This procedure is initiated in the point **202** from fig. 2 of the drawing. The current and auxiliary sector is selected on the basis of analysis of a few criteria.

The first is if previous data writing was correctly completed. The second is if the sector contains data compliant with the required format. The third of the possibilities appears in case when emulation of the EEPROM memory is initiated for the first time.

The procedure starts operation in the point **901**, where the first sector is set as the current one. Next, in the point **902** of the procedure it is checked if the saved data have a correct format and if they were correctly saved. It can be determined by analyzing appropriate bits of the patch header. If the check from step **903** determi-

nes that there are incorrect data in the first sector, the second sector is set as the current sector, and the first one as the auxiliary **909**. If the check from the step **903** determines that there are correct data in the first sector, the second sector is set as the current one in the point **904**. Next, in the point **905** of the procedure it is checked if the saved data have a correct format and if the data were correctly saved. If the check from the point **906** determines that there are incorrect data in the second sector, the first sector is set as the current sector, and the second one as the auxiliary **908**. If the check from step **906** determines that there are correct data in the first sector, in the point **907** of the procedure the sector in which there is more free space is set as the current sector. The sector, in which there is more free space, contains more current data. The procedure of selecting the current and auxiliary sector ends in the point **910**.